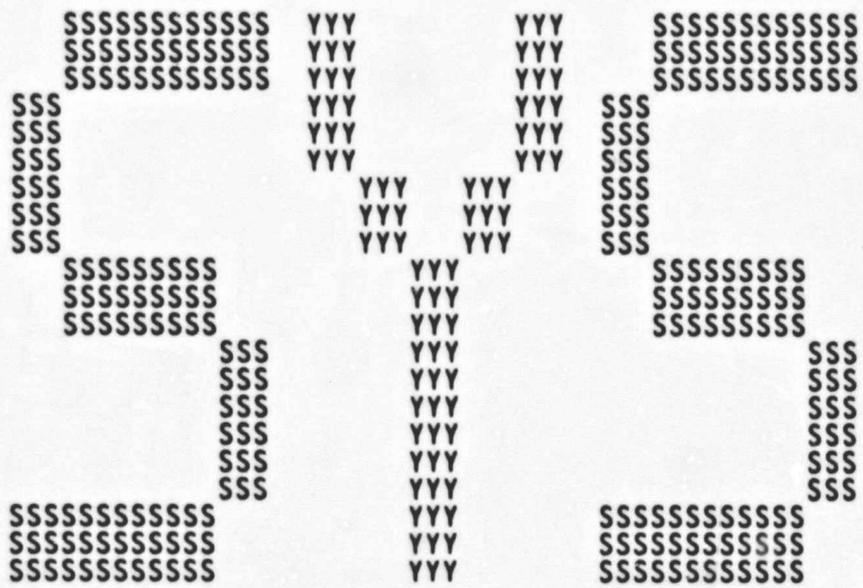
SSSSSSSSSSS YYY YYY SSSSSSSSSSS
SSSSSSSSSSS YYY YYY SSSSSSSSSSS
SSSSSSSSSSS YYY YYY SSSSSSSSSSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSSSSSSSS YYY SSSSSSSSS
SSSSSSSSS YYY SSSSSSSSS
SSSSSSSSS YYY SSSSSSSSS
SSS YYY SSS
SSS YYY SSS
SSS YYY SSS
SSS YYY SSS
SSS YYY SSS
SSS YYY SSS
SSSSSSSSSSS YYY SSSSSSSSSSS
SSSSSSSSSSS YYY SSSSSSSSSSS
SSSSSSSSSSS YYY SSSSSSSSSSS

FILEID**SYSMAR

```
  SSSSSSSS  YY      YY   SSSSSSSS  MM      MM    AAAAAA    RRRRRRRR
  SSSSSSSS  YY      YY   SSSSSSSS  MM      MM    AAAAAA    RRRRRRRR
SS          YY      YY  SS         MMMM  MMMM  AA      AA  RR      RR
SS          YY      YY  SS         MMMM  MMMM  AA      AA  RR      RR
SS            YY  YY    SS         MM  MM  MM  AA      AA  RR      RR
SS            YY  YY    SS         MM  MM  MM  AA      AA  RR      RR
  SSSSSS        YY      SSSSSS     MM      MM  AA      AA  RRRRRRRR
  SSSSSS        YY      SSSSSS     MM      MM  AA      AA  RRRRRRRR
        SS      YY            SS   MM      MM  AAAAAAAAAA  RR  RR
        SS      YY            SS   MM      MM  AAAAAAAAAA  RR  RR
        SS      YY            SS   MM      MM  AA      AA  RR    RR      ....
        SS      YY            SS   MM      MM  AA      AA  RR    RR      ....
SSSSSSSS        YY    SSSSSSSS     MM      MM  AA      AA  RR      RR    ....
SSSSSSSS        YY    SSSSSSSS     MM      MM  AA      AA  RR      RR    ....
```

```
MM      MM    AAAAAA    RRRRRRRR
MM      MM    AAAAAA    RRRRRRRR
MMMM  MMMM  AA      AA  RR      RR
MMMM  MMMM  AA      AA  RR      RR
MM  MM  MM  AA      AA  RR      RR
MM  MM  MM  AA      AA  RR      RR
MM      MM  AA      AA  RRRRRRRR
MM      MM  AA      AA  RRRRRRRR
MM      MM  AAAAAAAAAA  RR  RR
MM      MM  AAAAAAAAAA  RR  RR
MM      MM  AA      AA  RR    RR
MM      MM  AA      AA  RR    RR
MM      MM  AA      AA  RR      RR
MM      MM  AA      AA  RR      RR
```

```
        .NLIST
;
; Version:      'V04-000'
;
        .LIST ME
;
;*****************************************************************************
;*                                                                           *
;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
;*  ALL RIGHTS RESERVED.                                                     *
;*                                                                           *
;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
;*  TRANSFERRED.                                                             *
;*                                                                           *
;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
;*  CORPORATION.                                                             *
;*                                                                           *
;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
;*                                                                           *
;*                                                                           *
;*****************************************************************************
;
;++
; FACILITY:  VAX/VMS System Macro Libraries
;
; ABSTRACT:
;
;       This file contains macros that are commonly used by the
;       Executive and drivers.
;
; ENVIRONMENT:
;
;       n/a
;--
;
; AUTHOR:  The VMS Group          CREATION DATE:  1-Aug-1976
;
; MODIFIED BY:
;
;       V03-025 RLRADAPD        Robert L. Rappaport     15-Mar-1984
;               Move ADAPDESC macro here from INIADP.MAR.  Also add
;               ADAP_INIRUT macro definition.
;
;       V03-024 WHM0001         Bill Matthews           05-Mar-1984
;               Add support to SLVTAB macro for specifying the address of
;               the vectors in SYS.EXE.
```

```
;
;       V03-023 TMK0002         Todd M. Katz            13-Feb-1984
;               Modify REQUEST_DATA and SEND_DATA so that if the fork process
;               call to FPC$ALLOCMSG returns an error, the fork process call
;               to FPC$REQDATA and FPC$SENDDATA respectively is skipped.
;
;       V03-022 TMK0001         Todd M. Katz            06-Feb-1984
;               Add SEND_DG_BUF_REG for sending a datagram buffer without
;               having a CDRP by calling FPC$SENDRGDG with all the registers
;               for sending the datagram already initialized with the
;               appropriate values.
;
;       V03-021 ROW0289         Ralph O. Weber          26-JAN-1984
;               Add three DDTAB parameters for the various driver-specific
;               flavors of mount verification:  MNTV_SQD for sequential device
;               mount verification, MNTV_FOR for foreign device mount
;               verification, and MNTV_SSSC for shadow set state change mount
;               verification.
;
;       V03-020 TCM0003         Trudy C. Matthews       02-Aug-1983
;               Update CPUDISP macro so that it correctly handles the
;               11/785 format System Identification register.
;
;       V03-019 KDM0047         Kathleen D. Morse       07-Jun-1983
;               Added TIMEDWAIT macro, which will eventually replace
;               TIMEWAIT because its parameters are too restrictive
;               for all environments.
;
;       V03-018 RLRCPUDISPa     Robert L. Rappaport     15-Jun-1983
;               Add ENVIRON argument to CPUDISP so as to conditionally
;               generate a BUG_CHECK where appropriate.
;
;       V03-017 WMC0001         Wayne Cardoza           29-May-1983
;               Add more protection arguments to SLVTAB.
;
;       V03-016 RLRTEMP         Robert L. Rappaport     31-May-1983
;               Temporary fix to CPUDISP so as to procede with build.
;               Later fix will add ENVRION argument to CPUDISP.
;
;       V03-015 DWT0101         David W. Thiel          25-May-1983
;               Add IFCLSTR and IFNOCLSTR macros which determine
;               whether or not a system is in a cluster environment.
;
;       V03-014 RLRCPUDISP      Robert L. Rappaport     25-May-1983
;               Have CPUDISP use DISPATCH macro rather than the CASE
;               macro.  Do this in a way that for now we will accept
;               both formats of CPUDISP.  Later when all CPUDISP's
;               have been recoded, we will reject old style invocations.
;
;       V03-013 JWH0213         Jeffrey W. Horn         13-Apr-1983
;               Change SLVTAB so that it can be used more than once
;               per module.
;
;       V03-012 ROW0176         Ralph O. Weber          4-APR-1983
;               Add macro for the fork-and-wait executive service, FORK_WAIT.
```

```
;       V03-011 ACG0322         Andrew C. Goldstein,    25-Mar-1983  13:21
;               Change IFPRIV and IFNPRIV to use privilege mask in PCB
;
;       V03-010 JWH0202         Jeffrey W. Horn         22-Mar-1983
;               Add SLVTAB macro.  Also add additional LOADVEC types.
;
;       V03-009 MSH0001         Maryann Hinden          25-Feb-1983
;               Delete .EXTERNAL definitions for SCS macros.
;
;       V03-008 SRB0060         Steve Beckhardt         6-Jan-1983
;               Added DISPATCH macro.
;
;       V03-007 ROW0144         Ralph O. Weber          8-DEC-1982
;               Add the following SCS macros:
;                  o  RECYCL_RSPID recycle a response ID.
;                  o  FIND_RSPID_RDTE locate and validate the RDTE for a
;                     given response ID.
;                  o  SCAN_MSGBUF_WAIT like SCAN_RSPID_WAIT but scans message
;                     buffer and send credit wait queues.
;
;       V03-006 KTA3018         Kerbey T. Altmann       12-Nov-1982
;               Modify LOAVEC to add SEC_LABEL param.
;
;       V03-005 TCM0002         Trudy C. Matthews       12-Oct-1982
;               Change TIMEWAIT macro to use a SOBGTR loop to introduce a
;               delay into its bit test loop (instead of NOPs).
;
;       V03-004 STJ3027         Steven T. Jeffreys      24-Sep-1982
;               Add the LOADVEC macro.
;
;       V03-003 ROW0125         Ralph O. Weber          19-SEP-1982
;               Add the CLONEDUCB argument to the DDTAB macro.
;
;       V03-002 RAS0095         Ron Schaefer            30-Aug-1982
;               Change the CASE macro to generate signed offsets so
;               the linker can report truncation errors.
;
;**
;
; GENERATING SYSTEM INTERNAL BUG CHECK
;
; BUG_CHECK ERROR,TYPE
;
;       ERROR  = ONE TO SIX CHARACTER ERROR NAME.
;       TYPE   = "FATAL" OR ANYTHING ELSE.
;
;
        .MACRO  BUG_CHECK ERROR,TYPE=CONT
                .WORD   ^XFEFF
                .IIF IDN <TYPE>,<FATAL> , .WORD BUG$_'ERROR'!4
                .IIF DIF <TYPE>,<FATAL> , .WORD BUG$_'ERROR'
        .ENDM   BUG_CHECK
;
; GENERATE OPERATING BUG CHECK
;
```

```
; BUGCHK SUBSYSTEM,ERROR,MODE [,CALLOP=JSB]
;
        .MACRO  BUGCHK SUBSYSTEM,ERROR,MODE,CALLOP=BSBW
        .IF IDN <MODE>,<FATAL>
        CALLOP  EXE$BUGCHKFATAL
        .IFF
        CALLOP  EXE$BUGCHKCONT
        .ENDC
        .ASCIZ  /SUBSYSTEM'_'ERROR/
        .ENDM   BUGCHK

;
;       CASE MACRO FOR GENERATING CASE AND CASE TABLE
;
;       CASE    SRC,<DISPATCH LIST>,[TYPE=W/B/L],[LIMIT=#0],[NMODE=S^#]
;
        .MACRO  CASE,SRC,DISPLIST,TYPE=W,LIMIT=#0,NMODE=S^#,?BASE,?MAX
        CASE'TYPE       SRC,LIMIT,NMODE'<<MAX-BASE>/2>-1
BASE:
        .IRP    EP,<DISPLIST>
        .SIGNED_WORD    EP-BASE
        .ENDR
MAX:
        .ENDM

;
; DISPATCH -- Dispatch on set of index values, not necessarily dense.
;
; This macro translates into the CASEx instruction.  It calculates the
; "base" and "limit" parameters from the <index,displacement> list
; specified in the 'veclist' parameter.  The dispatch table is set up
; such that any unspecified index value within the bounds of the
; transfer vector is associated with a displacement which transfers
; control to the first location after the CASE statement, i.e., behaves
; as if the index were out of bounds.
;
; Note that since the index values themselves appear in the vector
; (presumably symbolically), no ASSUME statements are needed.
;
; The prefix argument may be used to specify a common symbolic prefix
; for all the index values.
;
; This macro works as follows:
;       $$MAX and $$MIN are macros used to find the highest and lowest
;               index value
;       $$GENDISPL is a macro used to generate a displacement if the
;               appropriate index value is specified in the vector list
;
;       First the maximum and minimum index values are found from
;       which the base and limit operands may be calculated and the
;       instruction generated.
;       Then, $$GENDISPL is called for each index value in range to
;       generate a branch displacement if the appropriate value was
;       specified.  If it wasn't, then a branch displacement is generated
;       to the "fall through" point.
```



```
;
;+
; L
; I
;
;
;
; O
;
;
;
;
;
;
;
;
;-

TRY

ERR
EXI

;+
; U
; I
;
;
;
;
; O
;
;-

EXI
```

```
;
; NOTE: This macro assembles in "N squared" time where N is the size (limit)
;       of the CASE.  There are other approaches to doing this macro that
;       will assemble in "linear with N" time.  If the inefficiency of this
;       approach is a problem for you, please feel free to rewrite it.
;

        .MACRO  DISPATCH        INDEX,VECLIST,TYPE=W,PREFIX=<>,?DISPL0

        .MACRO  $$MAX   NUM,IGNORE
        .IIF    EQ $$MXSW, $$HIGH=NUM
        $$MXSW=1
        .IIF    LT $$HIGH-NUM, $$HIGH=NUM
        .ENDM   $$MAX

        .MACRO  $$MIN   NUM,IGNORE
        .IIF    EQ $$MNSW, $$LOW=NUM
        $$MNSW=1
        .IIF    GT  $$LOW-NUM, $$LOW=NUM
        .ENDM   $$MIN

        .MACRO  $$GENDISPL      VALUE,LABEL
        .IF     EQ  $$DISPL-VALUE
        .SIGNED_WORD    LABEL-DISPL0
        .IIF    EQ  1-$$GENSW,  .ERROR  ; Duplicate occurrence of VALUE ;
        $$GENSW=1
        .ENDC
        .ENDM   $$GENDISPL

$$MXSW=0
$$MNSW=0
        .IRP    TUPLE,<VECLIST>
        $$MAX   PREFIX''TUPLE
        $$MIN   PREFIX''TUPLE
        .ENDR
$$BASE=$$LOW
$$LIMIT=$$HIGH-$$LOW
$$DISPL=$$BASE
        CASE'TYPE       INDEX,#$$BASE,#$$LIMIT
DISPL0:
        .REPT   $$LIMIT+1
        $$GENSW=0
        .IRP    TUPLE,<VECLIST>
        $$GENDISPL      PREFIX''TUPLE
        .ENDR
        .IIF    EQ  $$GENSW,    .WORD   2*<$$LIMIT+1>
        $$DISPL=$$DISPL+1
        .ENDR
        .ENDM   DISPATCH


;
; CPU TYPE DISPATCH MACRO:
;
;       CPUDISP IS INVOKED TO HANDLE CPU DIFFERENCES IN LINE, E.G.,
;
```

```
;               CPUDISP <<780,10$>,<750,20$>,<730,30$>,<790,40$>>
                                        ;*DISPATCH ON CPU TYPE*
;       10$:    <11/780 SPECIFIC CODE>
;       20$:    <11/750 SPECIFIC CODE>
;       30$:    <11/730 SPECIFIC CODE>
;       40$:    <11/790 SPECIFIC CODE>
                                        ;*END OF CPU-DEPENDENT CODE*
;
;       THE CPUDISP MACRO IS INVOKED WITH A LIST OF PAIRS (2-TUPLES)
;               WHEREIN THE FIRST ELEMENT OF EACH PAIR IS THE PROCESSOR
;               TYPE (E.G. 780, 750, ETC.) AND THE SECOND ELEMENT IS
;               THE ADDRESS OF WHERE CODE SPECIFIC TO THAT CPU TYPE
;               IS LOCATED.
;       THIS MACRO, THROUGH ITS INVOCATION OF THE DISPATCH MACRO, RESULTS
;               IN A CASEB INSTRUCTION AND ITS DISPATCH LIST.
;       THE ORDER OF SPECIFICATION OF THE PAIRS IS NOT IMPORTANT AND
;               ANY HOLES IN THE SPECIFICATION LIST WILL RESULT IN
;               TRANSFERS TO THE CODE FOLLOWING THE DISPATCH LIST
;               WHERE A BUG_CHECK IS LOCATED.  THIS WILL PREVENT
;               INADVERTANT OMMISSIONS FROM PASSING UNNOTICED.
;       AS NEW CPU'S ARE ADDED, ALL OCCURRENCES OF CPUDISP MUST BE
;               EXPANDED TO HANDLE THEM.
;       THIS MACRO DEPENDS ON THE FACT THAT THE PROCESSOR TYPES ARE
;               SYMBOLICALLY SPECIFIED BY SYMBOLS OF THE FORM:
;
;                       PR$_SID_TYPxxx
;
;               WHERE xxx = 780, OR 750, OR 730, OR 790, ETC.
;
;       THE CPUDISP ALSO TAKES AN OPTIONAL ARGUMENT, ENVIRON, WHICH
;       DESCRIBES THE RUNTIME ENVIRONMENT.  ENVIRON=VMS IMPLIES NORMAL
;       SYSTEM RUNNING TIME.  IF THIS VALUE IS SPECIFIED THEN A BUG_CHECK
;       INVOCATION IS CODED IMMEDIATELY FOLLOWING THE DISPATCH LIST SO
;       THAT FAILURE TO PROVIDE THE PROPER CPU TYPE, WILL RESULT IN
;       A BUG_CHECK AT RUNTIME.  IF ENVIRON=VMB IS CODED, THEN THE EQUIVALENT
;       OF A BUG_CHECK AT VMB TIME, I.E. A BSBW TO ERROUT SPECIFYING AN
;       AN UNKNOWN PROCESSOR TYPE IS GENERATED.

        .MACRO  CPUDISP,ADDRLIST,ENVIRON=VMS,?Z

;
; This internal macro tests to see if a destination was specified for the
; 11/785.  (Usually, the 785 processor will execute the same code path as
; the 11/780, as their CPU type fields in the SID are identical.)  If so,
; see if this CPU is an 11/785.  If so, branch directly to the 11/785
; destination (i.e. skip over the CASE instruction).
;
        .MACRO  TEST785 CPU,DEST,?LBL
        .IF     IDN     <CPU>,<785>
        PR$_SID_TYP785 = PR$_SID_TYPMAX + 1
                                        ; DISPATCH macro needs this definition.
        CMPB    G^EXE$GB_CPUTYPE, -     ; Is this an 11/785?
                #PR$_SID_TYP780
        BNEQ    LBL                     ; Branch if not.
        BBC     #23,G^EXE$GB_CPUDATA,LBL; Branch if not.
        BRW     DEST                    ; Branch to execute 11/785 code.
```

```
LBL:
        .ENDC
        .ENDM   TEST785

;
; This internal macro tests to see which format of CPUDISP is being invoked.
;
        .MACRO  TESTARGS,ARG1,ARG2,?Q
        .NCHR   Q,<ARG2>
        .IIF    GT      Q,      Z=1
        .ENDM   TESTARGS

Z=0
        .IRP    D,<ADDRLIST>
        TESTARGS        D
        TEST785         D
        .ENDR
        .IF     NE      Z
        DISPATCH        G^EXE$GB_CPUTYPE,<ADDRLIST>,TYPE=B,PREFIX=PR$_SID_TYP
        .IFF
        CASE    G^EXE$GB_CPUTYPE,<ADDRLIST>,LIMIT=#PR$_SID_TYP780,TYPE=B
        .ENDC

        .IF     IDN     <ENVIRON>,<VMS>
        BUG_CHECK       UNSUPRTCPU,FATAL
        .IFF
                .IF     IDN     <ENVIRON>,<VMB>
                BSBW    ERROUT
                .ASCIZ  /%BOOT-F-Unknown processor/
                .IFF
                        .IF     IDN     <ENVIRON>,<XDELTA>
                        HALT
                        .ENDC
                .ENDC
        .ENDC


        .ENDM   CPUDISP
;
; GENERATE DRIVER DISPATCH TABLE
;
; DDTAB DEVNAM,START,UNSOLIC,FUNCTB,CANCEL,REGDMP,DIAGBF,ERLGBF,UNITINIT, -
;       ALTSTART,MNTVER,CLONEDUCB,MNTV_SSSC,MNTV_FOR,MNTV_SQD
;
; FDTSIZE is defined by FUNCTAB macro, it is zeroed here as well so a driver
; can have a DDTAB without a FUNCTAB. It is not done here with a
; .IF NOT_DEFINED macro as MACRO will then immediately store the zero (on
; the first pass), and the value calculated by the FUNCTAB macro will
; be ignored.
;
        .MACRO  DDTAB   DEVNAM,-
                        START=0,-
                        UNSOLIC=0,-
                        FUNCTB,-
                        CANCEL=0,-
                        REGDMP=0,-
```

```
                        DIAGBF=0,-
                        ERLGBF=0,-
                        UNITINIT=0,-
                        ALTSTART=0,-
                        MNTVER=+IOC$MNTVER,-
                        CLONEDUCB=0,-
                        MNTV_SSSC=0,-
                        MNTV_FOR=0,-
                        MNTV_SQD=0
        .PSECT  $$$115_DRIVER,LONG
'DEVNAM'$DDT::
        GENRADDR START,'DEVNAM'$DDT
        GENRADDR UNSOLIC,'DEVNAM'$DDT
        GENRADDR FUNCTB,'DEVNAM'$DDT
        GENRADDR CANCEL,'DEVNAM'$DDT
        GENRADDR REGDMP,'DEVNAM'$DDT
        .WORD   DIAGBF
        .WORD   ERLGBF
        GENRADDR UNITINIT,'DEVNAM'$DDT
        GENRADDR ALTSTART,'DEVNAM'$DDT
        GENRADDR MNTVER,'DEVNAM'$DDT
        GENRADDR CLONEDUCB,'DEVNAM'$DDT
        .WORD   FUNCTAB_LEN, 0
        GENRADDR MNTV_SSSC,'DEVNAM'$DDT
        GENRADDR MNTV_FOR,'DEVNAM'$DDT
        GENRADDR MNTV_SQD,'DEVNAM'$DDT
FUNCTAB_LEN = 0
        .ENDM   DDTAB

;
; DECREMENT PAGE REFERENCE COUNT
;
        DECREF  EQL,GTR,PFN,MODE,LABEL,CALL

        EQL     = BRANCH LOCATION IF NEW REFCNT = 0
        GTR     = BRANCH LOCATION IF NEW REFCNT > 0
        PFN     = REGISTER CONTAINING PFN, DEFAULT TO R0
        MODE    = ADDRESSING MODE, DEFAULT IS WORD DISPLACEMENT
        LABEL   = IF PRESENT, USE THE SUPPLIED PARAMETER AS A LABEL.
                  OTHERWISE CREATE A LOCAL LABEL.
        CALL    = IF PRESENT, USE A JSB TO CALL MMG$REFCNTNEG.
                  OTHERWISE USE A BSBW.


        .MACRO  DECREF  EQL,GTR,PFN=R0,MODE=W^,?L1,CALL
        TMP...=0
                DECW    @'MODE'PFN$AW_REFCNT[PFN]
        .IF     NB,EQL
                BEQL    EQL
                TMP...=TMP...+1
        .ENDC
        .IF     NB,GTR
                BGTR    GTR
                TMP...=TMP...+1
        .ENDC
        .IF     LT,<TMP...-2>
```

```
                BGEQ    L1
        .ENDC
        .IF     NB,CALL
                JSB     G^MMG$REFCNTNEG
        .IFF
                BSBW    MMG$REFCNTNEG
        .ENDC
        .IF     LT,<TMP...-2>
L1:
        .ENDC
        .ENDM   DECREF
;
; DECREMENT PAGE SHARE COUNT
;
        DECSHR  EQL,GTR,PFN,IMAGE_FLAG

        EQL = BRANCH LOCATION IF NEW SHRCNT = 0
        GTR = BRANCH LOCATION IF NEW SHRCNT > 0
        PFN = REGISTER CONTAINING PFN, DEFAULT TO R0
        IMAGE_FLAG = Indicator of whether macro is located in nonpaged exec
                        Set to SYS_NONPAGED if so
                        Defaults to NOSYS

        .MACRO  DECSHR  EQL,GTR,PFN=R0,IMAGE_FLAG=NOSYS,?L1
        TMP...=0
                        PFN_REFERENCE   -
        DECW    <@W^PFN$Ax_SHRCNT[PFN]>,-
                        LONG_OPCODE=DECL,-
                        IMAGE=IMAGE_FLAG
        .IF     NB,EQL
                BEQL    EQL
                TMP...=TMP...+1
        .ENDC
        .IF     NB,GTR
                BGTR    GTR
                TMP...=TMP...+1
        .ENDC
        .IF     LT,<TMP...-2>
                BGEQ    L1
        .ENDC
                BSBW    MMG$SHRCNTNEG
        .IF     LT,<TMP...-2>
L1:
        .ENDC
        .ENDM   DECSHR
;
; DEVICE DRIVER PROLOGUE TABLE
;
        DPTAB   END,ADAPTER,FLAGS,UCBSIZE,UNLOAD,MAXUNITS,DEFUNITS,DELIVER
                NAME


        END =   ADDR OF END OF DRIVER
        ADAPTER = ADAPTER TYPE (UBA,MBA,DRA)
        FLAGS = DRIVER LOADER FLAGS
        UCBSIZE = SIZE OF EACH UCB (IN BYTES)
```

```
;       UNLOAD = ADDRESS OF A ROUTINE TO CALL IF THE DRIVER IS TO BE UNLOADED
;       MAXUNITS = MAXIMUM NUMBER OF UNITS THAT CAN BE CONNECTED.
;       DEFUNITS = DEFAULT NUMBER OF UNITS TO AUTOCONFIGURE.
;       DELIVER = ADDRESS OF A ROUTINE TO CALL AT AUTOCONFIGURE TO DELIVER UNITS
;       VECTOR = OFFSET TO SET OF VECTORS (USED BY TTDRIVER)
;       NAME = DRIVER NAME
;
;
        .MACRO  DPTAB   END,ADAPTER,FLAGS=0,UCBSIZE,UNLOAD,MAXUNITS=8,-
                        DEFUNITS=1,DELIVER,VECTOR,NAME
        .SAVE
        $DPTDEF
        ASSUME  DPT$C_LENGTH EQ 56
        .PSECT  $$$105_PROLOGUE
DPT$TAB:
        .BLKL   2

        .WORD   END-DPT$TAB
        .BYTE   DYN$C_DPT
        .BYTE   0

        .BYTE   AT$_'ADAPTER'
        .BYTE   FLAGS
        .WORD   UCBSIZE

        .WORD   DPT$INITAB-DPT$TAB
        .WORD   DPT$REINITAB-DPT$TAB

        .IF NB,UNLOAD
        .WORD   UNLOAD-DPT$TAB
        .IFF
        .WORD   0
        .ENDC

        .WORD   MAXUNITS
        .WORD   DPT$C_VERSION
        .WORD   DEFUNITS

        .IF NB,DELIVER
        .WORD   DELIVER-DPT$TAB
        .IFF
        .WORD   0
        .ENDC

        .IF NB,VECTOR
        .WORD   VECTOR-DPT$TAB
        .IFF
        .WORD   0
        .ENDC

        $$$=.
        .ASCIC  /NAME/
        .=$$$+12

        .LONG   0,0             ; LINK TIME
        .LONG   0               ; ECO LEVEL
```

```
        .MDELETE DPTAB
        .ENDM   DPTAB

;
; STORE DPT INITIALIZATION TABLE VALUES
;
        DPT_STORE STRUC_TYPE,STRUC_OFFSET,OPERATION,EXPRESSION,POS,SIZE

        STRUC_TYPE = STRUC TYPE CODE (DDB,UCB,CRB,IDB)
                   = "INIT" IF START OF INIT TABLE
                   = "REINIT" IF START OF RE-INIT TABLE
                   = "END" IF END OF RE-INIT TABLE
        STRUC_OFFSET = UNSIGNED OFFSET INTO STRUC
        OPERATION = TYPE OF INITIALIZATION OPERATION
                    B=BYTE,W=WORD,L=LONG,D=RELATIVE TO DRIVER,V=FIELD
                    IF PRECEDED BY "@" (IE. @B) THEN EXPRESSION
                    IS ADDRESS OF DATA
        EXPRESSION = EXPRESSION TO BE STORED
        POS = BIT POSITION (FOR OPERATION = V ONLY)
        SIZE = FIELD SIZE (FOR OPERATION = V ONLY)
;
        .MACRO  DPT_STORE STR_TYPE,STR_OFF,OPER,EXP,POS,SIZE
        .IF IDN <STR_TYPE>,<INIT>
DPT$INITAB:
        .IFF
        .IF IDN <STR_TYPE>,<REINIT>
DPT$REINITAB:
        .IFF
        .IF IDN <STR_TYPE>,<END>
        .BYTE   0
        .RESTORE

        .IFF
        .BYTE   DYN$C_'STR_TYPE
        .BYTE   STR_OFF
        $$OP=0
        .IRPC   C,<OPER>
        .IIF IDN <C>,<@>, $$OP=^X80
        .IIF IDN <C>,<W>, $$OP=$$OP!1
        .IIF IDN <C>,<D>, $$OP=$$OP!2
        .IIF IDN <C>,<L>, $$OP=$$OP!3
        .IIF IDN <C>,<V>, $$OP=$$OP!4
        .ENDR
        .BYTE   $$OP
        .IF EQ  $$OP
        .BYTE   EXP
        .IFF
        .IF EQ $$OP-1
        .WORD   EXP
        .IFF
        .IF EQ $$OP-2
        .WORD   EXP-DPT$TAB
        .IFF
        .LONG   EXP
        .IIF NB,POS, .BYTE POS
```

```
        .IIF NB,SIZE, .BYTE SIZE
        .ENDC
        .ENDC
        .ENDC


        .ENDC
        .ENDC
        .ENDC


        .ENDM   DPT_STORE
;
; DISABLE INTERRUPTS
;
; DSBINT IPL,DST
;

        .MACRO  DSBINT IPL,DST
                .IF B   DST
                MFPR    S^#PR$_IPL,-(SP)
                .IFF
                MFPR    S^#PR$_IPL,DST
                .ENDC
                .IF B   IPL
                MTPR    #31,S^#PR$_IPL
                .IFF
                MTPR    IPL,S^#PR$_IPL
                .ENDC
        .ENDM   DSBINT

;
; ENABLE INTERRUPTS
;
; ENBINT SRC
;

        .MACRO  ENBINT SRC
                .IF B   SRC
                MTPR    (SP)+,S^#PR$_IPL
                .IFF
                MTPR    SRC,S^#PR$_IPL
                .ENDC
        .ENDM   ENBINT

;
; MACRO TO DEFINE SOME OF THE ERROR MESSAGE BUFFER OFFSET VALUES
;
; CALL:           $EMBDEF <LIST>
;
; WHERE:          LIST IS A SERIES OF 2 CHARACTER CODES FOR THE TYPE
;                 OF ERROR MESSAGES THE OFFSETS ARE DESIRED
;
;       EG:       $EMBDEF <BC,CR,DV>
;
;       WOULD DEFINE CODES FOR BUGCHECK,CRASH, AND DEVICE ERROR MESSAGES.
```

```
;
        .MACRO  $EMBDEF LIST=<DV,TS>
        $EMBETDEF                       ; DEFINE ENTRY TYPE CODES
        $EMBHDDEF                       ; DEFINE HEADER OFFSETS
        .IRP    Z,<LIST>
        $EMB'Z'DEF
        .ENDR
        .ENDM   $EMBDEF

;
; FUNCTION TABLE ENTRY MACRO
;
; FUNCTAB ACTION ROUTINE,FUNCTION CODES
;
; NULL ACTION ROUTINE DOES NOT EXPAND A ACTION ADDRESS
;
        .MACRO  FUNCTAB ACTION,CODES
MASKL   = 0
MASKH   = 0
        .IF NOT_DEFINED FUNCTAB_LEN
FUNCTAB_LEN = 0
        .ENDC

        .IRP    X,<CODES>
        .IF     GT      <IO$_'X&IO$_VIRTUAL>-31
MASKH   = MASKH!<1@<<IO$_'X&IO$_VIRTUAL>-32>>
        .IFF
MASKL   = MASKL!<1@<IO$_'X&IO$_VIRTUAL>>
        .ENDC
        .ENDM
        .LONG   MASKL,MASKH
        FUNCTAB_LEN = FUNCTAB_LEN + 8
        .IF NB ACTION
        GENRADDR ACTION,<.+8>
        FUNCTAB_LEN = FUNCTAB_LEN + 4
        .ENDC
        .ENDM

;
; GENERATE RELATIVE ADDRESS FOR DRIVER DISPATCH AND FUNCTION DECISION TABLES
;
; GENRADDR ADDRESS,BASE
;

        .MACRO  GENRADDR ADDRESS,BASE
        .IF IDN <ADDRESS>,<0>
        .LONG   IOC$RETURN
        .IFF
        .IRPC   X,<ADDRESS>
        .IF IDN <X>,<+>
        .LONG   ADDRESS
        .IFF
        .LONG   ADDRESS-BASE
        .ENDC
        .MEXIT
```

```
        .ENDM
        .ENDC
        .ENDM   GENRADDR

;
; TEST IF PROCESS HAS SPECIFIED PRIVILEGE AND BRANCH ON TRUE
;
; IFPRIV PRIV,DEST,PCBREG
;
        .MACRO  IFPRIV PRIV,DEST,PCBREG=R4
                .IF DIF <PRIV>,<R1>
                .IF DIF <PRIV>,<R2>
                BBS     #PRV$V_'PRIV,PCB$Q_PRIV(PCBREG),DEST
                .IFF
                BBS     PRIV,PCB$Q_PRIV(PCBREG),DEST
                .ENDC
                .IFF
                BBS     PRIV,PCB$Q_PRIV(PCBREG),DEST
                .ENDC
        .ENDM   IFPRIV

;
; TEST IF PROCESS DOES NOT HAVE PRIVILEGE AND BRANCH ON TRUE
;
; IFNPRIV PRIV,DEST,PCBREG
;
        .MACRO  IFNPRIV PRIV,DEST,PCBREG=R4
                .IF DIF <PRIV>,<R1>
                .IF DIF <PRIV>,<R2>
                BBC     #PRV$V_'PRIV,PCB$Q_PRIV(PCBREG),DEST
                .IFF
                BBC     PRIV,PCB$Q_PRIV(PCBREG),DEST
                .ENDC
                .IFF
                BBC     PRIV,PCB$Q_PRIV(PCBREG),DEST
                .ENDC
        .ENDM   IFNPRIV

;
; BRANCH IF RANGE OF ADDRESSES IS NOT READABLE
;
; IFNORD SIZ,ADR,DEST,MODE
;
        .MACRO  IFNORD  SIZ,ADR,DEST,MODE=#0
                PROBER  MODE,SIZ,ADR
                BEQL    DEST
        .ENDM   IFNORD

;
; BRANCH IF RANGE OF ADDRESSES IS READABLE
;
; IFRD SIZ,ADR,DEST,MODE
;
```

```
        .MACRO  IFRD SIZ,ADR,DEST,MODE=#0
                PROBER  MODE,SIZ,ADR
                BNEQ    DEST
        .ENDM   IFRD

;
; BRANCH IF RANGE OF ADDRESSES IS NOT WRITABLE
;
; IFNOWRT SIZ,ADR,DEST,MODE
;

        .MACRO  IFNOWRT SIZ,ADR,DEST,MODE=#0
                PROBEW  MODE,SIZ,ADR
                BEQL    DEST
        .ENDM   IFNOWRT

;
; BRANCH IF RANGE OF ADDRESS IS WRITABLE
;
; IFWRT SIZ,ADR,DEST,MODE
;

        .MACRO  IFWRT   SIZ,ADR,DEST,MODE=#0
                PROBEW  MODE,SIZ,ADR
                BNEQ    DEST
        .ENDM   IFWRT

;
; CREATE I/O DRIVER FORK PROCESS
;
; IOFORK
;

        .MACRO  IOFORK
                JSB     G^EXE$IOFORK
        .ENDM   IOFORK

;
; CREATE FORK PROCESS
;
; FORK
;

        .MACRO  FORK
                JSB     G^EXE$FORK
        .ENDM   FORK

;
; FORK AND WAIT (for from 0 to 1 seconds)
;
; FORK_WAIT
;

        .MACRO  FORK_WAIT
                JSB     G^EXE$FORK_WAIT
```

```
        .ENDM   FORK_WAIT

;
; INVALIDATE TRANSLATION BUFFER
; INVALID ADDR,REG
;
        .MACRO  INVALID ADDR,REG
                .IF B   ADDR
                MTPR    #0,S^#PR$_TBIA
                .IFF
                .IF B   REG
                MTPR    ADDR,S^#PR$_TBIS
                .IFF
                MOVL    ADDR,REG
                MTPR    REG,S^#PR$_TBIS
                .ENDC
                .ENDC
        .ENDM   INVALID

;
; LOAD P0 SPACE LENGTH REGISTER
;
        .MACRO  LDPOLR  SRC
        .LIST   MEB
                MTPR    SRC,S^#PR$_POLR
        .NLIST  MEB
        .ENDM   LDPOLR
;
; LOAD P1 SPACE LENGTH REGISTER
;
        .MACRO  LDP1LR  SRC
        .LIST   MEB
                MTPR    SRC,S^#PR$_P1LR
        .NLIST  MEB
        .ENDM   LDP1LR

;
; LOAD MBA MAP REGISTERS
;
        .MACRO  LOADMBA
                JSB     G^IOC$LOADMBAMAP
        .ENDM   LOADMBA

;
; LOAD UBA MAP REGISTERS
;
        .MACRO  LOADUBA
                JSB     G^IOC$LOADUBAMAP
        .ENDM   LOADUBA

;
; LOAD UBA MAP REGISTERS - ALTERNATE ENTRY POINT
```

```
;
        .MACRO  LOADUBAA
        JSB     G^IOC$LOADUBAMAPA
        .ENDM   LOADUBAA

;+
; LOCK - MACRO TO SET A LOCK BIT AND RETRY IF SET FAILS
;
; INPUTS:
;
;       FLAG = BIT POSITION TO SET
;       FIELD = BASE OF FIELD IN WHICH FLAG IS TO BE SET
;
; OUTPUTS:
;
;       R0 = SUCCESS IF FLAG CHANGED FROM CLEAR TO SET STATE IN
;            EXE$GL_LOCKRTRY RETRIES.
;          = FAILURE IF RETRIES EXCEEDED BEFORE FLAG'S STATE COULD
;            BE CHANGED.
;
;       IF SUCCESS:
;
;       (SP) = PREVIOUS IPL AND CURRENT IPL = 31.
;
;-
        .MACRO  LOCK    FLAG,FIELD,?EXIT,?ERROR,?TRY
        MOVL    G^EXE$GL_LOCKRTRY,R0
TRY:    DSBINT
        BBSSI   FLAG,FIELD,ERROR
        MOVL    #1,R0
        BRB     EXIT
ERROR:  ENBINT
        SOBGTR  R0,TRY
EXIT:
        .ENDM   LOCK

;+
; UNLOCK - MACRO TO CLEAR A LOCK BIT
;
; INPUTS:
;
;       FLAG = BIT POSITION TO CLEAR
;       FIELD = BASE OF FIELD IN WHICH FLAG IS TO BE CLEAR
;
;       (SP) = PREVIOUS IPL
;
; OUTPUTS:
;
;       FLAG CLEARED AND PREVIOUS IPL RESTORED.
;-
        .MACRO  UNLOCK  FLAG,FIELD,?EXIT

        BBCCI   FLAG,FIELD,EXIT
EXIT:
```

```
        ENBINT
        .ENDM   UNLOCK


;       .MACRO  PFN_DISP_IF_BIGPFN_THEN
;       .MACRO  PFN_DISP_ELSE
;       .MACRO  PFN_DISP_ENDIF
;
;  The following three macros provide a transparent method of making an
;  execution time decision on which code path to execute, depending on
;  the size of physical memory. This series of macros is provided for
;  the case where more than one instruction depends on physical memory
;  size. A single instruction that differs in more than the choice of
;  opcode must also use this macro. When a single instruction that differs
;  only in its opcode is the issue, the PFN_REFERENCE macro should be used.
;
;  The actual logical construction is as follows
;
;       PFN_DISP_IF_BIGPFN_THEN ; IF FLINK and BLINK are longword arrays THEN
;               block of code with longword references
;       PFN_DISP_ELSE           ; ELSE (if FLINK and BLINK are word arrays)
;               block of code with word references (This block is optional.)
;       PFN_DISP_ENDIF
;
;  These macros are currently implemented with byte branch displacements for
;  both the THEN and ELSE pieces. If necessary, the macros could be enhanced
;  to generate the correct branches when word displacements are required.

;       .MACRO  PFN_DISP_IF_BIGPFN_THEN         END_BIGPFN_CODE,MODE
;
;  The first argument to the PFN_DISP_IF_BIGPFN_THEN macro is the label
;  of the end of the block of code that executes in the event that more
;  than 32 Mbytes of physical memory is present (which implies that FLINK
;  and BLINK are longword arrays). This label may either locate a block
;  of code that executes in the event that the FLINK and BLINK arrays are
;  word arrays (IF-THEN-ELSE construction) or it may locate the end of code
;  that depends on the size of the PFN link arrays (IF-THEN construction).
;
;  The second argument allows an addressing mode other than general (G^)
;  to be selected in special cases where the linker's default selection
;  would be incorrect.

        .MACRO  PFN_DISP_IF_BIGPFN_THEN         END_BIGPFN_CODE=10$,MODE=G^
        TSTW    MODE'MMG$GW_BIGPFN
        BEQL    END_BIGPFN_CODE
.SHOW
        ;This code executes if the PFN link arrays are longword arrays.;
.NOSHOW
        .ENDM   PFN_DISP_IF_BIGPFN_THEN

;  The code that executes for large physical memory sizes is sandwiched
;  between the PFN_DISP_IF_BIGPFN_THEN macro and either a PFN_DISP_ELSE
;  macro or a PFN_DISP_ENDIF macro. This is the "then" half of the
;  conditional and contains longword references to the FLINK and BLINK
;  arrays.
```

```
;       .MACRO  PFN_DISP_ELSE   ELSE_CODE,COMMON_CODE
;
;  There are two parameters for this macro. The first parameter is the label
;  where the word code begins. The second parameter is the label where
;  PFN-dependent code ends and common code begins.

        .MACRO  PFN_DISP_ELSE   ELSE_CODE=10$,COMMON_CODE=20$
        BRB     COMMON_CODE
.SHOW
        ;This code executes if the PFN link arrays are word arrays.;
.NOSHOW
ELSE_CODE':
        .ENDM   PFN_DISP_ELSE

;  The code that executes for small physical memory sizes is sandwiched between
;  the PFN_DISP_ELSE and PFN_DISP_ENDIF macros. This is the "else" half of the
;  conditional and contains word references to the FLINK and BLINK arrays.

;       .MACRO  PFN_DISP_ENDIF  COMMON_CODE
;
;  The single parameter for this macro is the label where the two code
;  paths rejoin into a single code path. Note that the default parameters
;  to this series of macros assumes an IF-THEN-ELSE construction. If an
;  IF_THEN construction is used, an explicit parameter must be supplied
;  to the PFN_DISP_ELSE macro.

        .MACRO  PFN_DISP_ENDIF  COMMON_CODE=20$
.SHOW
        ;End of code that depends on size of PFN link arrays;
.NOSHOW
COMMON_CODE':
        .ENDM   PFN_DISP_ENDIF
;       .MACRO  PFN_REFERENCE
;
;  The PFN_REFERENCE macro replaces all single instruction references
;  to the PFN array elements whose size depends on physical memory size.
;  These arrays are
;
;       FLINK   Forward Link Array
;       BLINK   Backward Link Array
;       SHRCNT  Global Share Count Array (Overlays FLINK)
;       WSLX    Working Set List Index Array (Overlays BLINK)
;
;  The macro records the address of each such instruction, as well as the
;  opcode that must be used in the event that there is more than 32 Mbytes
;  of physical memory present. As a precautionary measure, a third table
;  contains the original opcode to allow verification while the fixups
;  are taking place. The address and opcode tables are used by
;  INIT to do bootstrap-time fixups in the event that there is more than
;  32 Mbytes present. If INIT detects that there is less than 32 Mbytes
;  present, nothing is done. That is, the default case is a system with
;  less than 32 Mbytes, with the relevant PFN array elements as words.
;
;  Note that opcode fixups can only be done on the nonpaged portion of
;  SYS.EXE. To allow for opcode selection in other places,
```

```
;  the macro also provides for an execution time decision in the event that
;  the instruction cannot be fixed up by INIT. This kind of decision must
;  be used by:
;
;       o  paged executive routines
;
;       o  dynamically loaded code (such as machine check handlers)
;
;       o  any external routine or image (including XDELTA)
;
;  The macro also provides for two-byte opcode because they are so easy
;  to include. This avoids lots of problems in the event that two-byte
;  opcodes are used by memory management in the future.
;
;  Parameters:
;
;       WORD_OPCODE     Opcode for word reference (inserted into SYS.EXE)
;
;       OPERANDS        Operands of instruction
;
;       LONG_OPCODE     Opcode for longword reference (stored in table)
;
;       IMAGE           Set to "SYS_NONPAGED" if INIT does opcode fixup. This
;                       setting should only be selected for references in the
;                       nonpaged portion of SYS.EXE.
;
;       MODE            Defaults to G^. This parameter can be set to @# or
;                       to L^ when the linker's default selection for G^
;                       addressing would be inappropriate, such as in
;                       module SHELL.
;
;  OPCODE_SIZE      Set to "TWO_BYTE" for two-byte opcode
;                   (The two-byte material in the macro is currently
;                    commented out because there is no example of a
;                    two-byte opcode reference to the PFN data base.)

        .MACRO  PFN_REFERENCE   -
                WORD_OPCODE,-
                OPERANDS,-
                LONG_OPCODE,-
                IMAGE=NOSYS,-
                MODE=G^,-
                OPCODE_SIZE=ONE_BYTE,-
                ?L_10$,?L_20$

        .IF     IDENTICAL       <IMAGE>,<SYS_NONPAGED>

        $OPDEF
        ...PFN = .
        .SAVE_PSECT     LOCAL_BLOCK
        .PSECT  Z$INIT$PFN_FIXUP_TABLE
;               .IF     IDENTICAL       <OPCODE_SIZE>,<TWO_BYTE>
;               .ADDRESS                ...PFN  ; Store data about low byte of opcode
;               .BYTE   <OP$_'WORD_OPCODE>&^X00FF
;               .BYTE   <OP$_'LONG_OPCODE>&^X00FF
;               .ADDRESS        ...PFN + 1 ; Store data about high byte of opcode
```

```
;               .BYTE   <<OP$_'WORD_OPCODE>&^XFF00>@-8
;               .BYTE   <<OP$_'LONG_OPCODE>&^XFF00>@-8
;               .IF_FALSE
        .ADDRESS        ...PFN
        .BYTE   OP$_'WORD_OPCODE
        .BYTE   OP$_'LONG_OPCODE
;               .ENDC
        .RESTORE_PSECT
        WORD_OPCODE     OPERANDS

        .IF_FALSE

        TSTW            MODE'MMG$GW_BIGPFN
        BNEQU           L_10$
        WORD_OPCODE     OPERANDS
        BRB             L_20$
L_10$:  LONG_OPCODE     OPERANDS
L_20$:

        .ENDC

        .ENDM   PFN_REFERENCE
;
; PURGE DATA PATH
;

        .MACRO  PURDPR
                JSB     G^IOC$PURGDATAP
        .ENDM   PURDPR

;+
; QRETRY - EXECUTE AN INTERLOCKED QUEUE INSTRUCTION AND RETRY IF FAILURE

; INPUTS:

        OPCODE = OPCODE NAME: INSQHI,INSQTI,REMQHI,REMQTI.
        OPERAND1 = FIRST OPERAND FOR OPCODE.
        OPERAND2 = SECOND OPERAND FOR OPCODE.
        SUCCESS = LABEL TO BRANCH TO IF OPERATION SUCCEEDS.
                  IF NOT SPECIFIED, MACRO JUST FALLS THRU.
        ERROR = LABEL TO BRANCH TO IF OPERATION FAILS.
                IF NOT SPECIFIED, MACRO JUST FALLS THRU.

; OUTPUTS:

        R0 = DESTROYED.

        C-BIT = CLEAR IF OPERATION SUCCEEDED.
                SET IF OPERATION FAILED - QUEUE LOCKED.
                (MUST BE CHECKED BEFORE V-BIT OR Z-BIT)

        REMQTI OR REMQHI -

                V-BIT = CLEAR IF AN ENTRY REMOVED FROM QUEUE.
                        SET IF NO ENTRY REMOVED FROM QUEUE.
```

```
;       INSQTI OR INSQHI -
;
;               Z-BIT = CLEAR IF ENTRY IS NOT FIRST IN QUEUE.
;                       SET IF ENTRY IS FIRST IN QUEUE.
;-
        .MACRO  QRETRY  OPCODE,OPERAND1,OPERAND2,SUCCESS,ERROR,?LOOP,?OK
        CLRL    R0
LOOP:
        OPCODE  OPERAND1,OPERAND2
        .IF NB SUCCESS
        BCC     SUCCESS
        .IFF
        BCC     OK
        .ENDC
        AOBLSS  G^EXE$GL_LOCKRTRY,R0,LOOP
        .IF NB ERROR
        BRB     ERROR
        .ENDC
OK:
        .ENDM   QRETRY


;
; I/O REQUEST COMPLETE
;
; REQCOM
;

        .MACRO  REQCOM
                JMP     G^IOC$REQCOM
        .ENDM   REQCOM

;
; RELEASE ALL CHANNELS
;
; RELCHAN
;

        .MACRO  RELCHAN
                JSB     G^IOC$RELCHAN
        .ENDM   RELCHAN

;
; RELEASE SECONDARY CHANNEL
;
; RELSCHAN
;

        .MACRO  RELSCHAN
                JSB     G^IOC$RELSCHAN
        .ENDM   RELSCHAN

;
; RELEASE UNIBUS DATAPATH
;
```

```
        .MACRO  RELDPR
                JSB     G^IOC$RELDATAP
        .ENDM   RELDPR

;
; RELEASE UNIBUS MAP REGISTERS
;

        .MACRO  RELMPR
                JSB     G^IOC$RELMAPREG
        .ENDM   RELMPR

;
; REQUEST PRIMARY CHANNEL
;
; REQPCHAN PRI
;

        .MACRO  REQPCHAN PRI
                .IF NB  PRI
                .IF IDN <HIGH>,<PRI>
                JSB     G^IOC$REQPCHANH
                .IFF
                JSB     G^IOC$REQPCHANL
                .ENDC
                .IFF
                JSB     G^IOC$REQPCHANL
                .ENDC
        .ENDM   REQPCHAN

;
; REQUEST SECONDARY CHANNEL
;
; REQSCHAN PRI
;

        .MACRO  REQSCHAN PRI
                .IF NB  PRI
                .IF IDN <HIGH>,<PRI>
                JSB     G^IOC$REQSCHANH
                .IFF
                JSB     G^IOC$REQSCHANL
                .ENDC
                .IFF
                JSB     G^IOC$REQSCHANL
                .ENDC
        .ENDM   REQSCHAN

;
; REQUEST UNIBUS DATAPATH
;

        .MACRO  REQDPR
                JSB     G^IOC$REQDATAP
        .ENDM   REQDPR
```

```
;
; REQUEST UNIBUS MAP REGISTERS
;

        .MACRO  REQMPR
                JSB     G^IOC$REQMAPREG
        .ENDM   REQMPR

;
; REPORT SYSTEM EVENT
;
; RPTEVT EVENTNAME
;

        .MACRO  RPTEVT,NAME,CALL_TYPE=BSB
                .IF     IDENTICAL       <CALL_TYPE>,<JSB>
                JSB     G^SCH$RSE
                .IF_FALSE
                BSBW    SCH$RSE
                .ENDC
                .BYTE   EVT$_'NAME
        .ENDM   RPTEVT

; SAVE PROCESSOR INTERRUPT PRIORITY LEVEL
;
; SAVIPL DST
;

        .MACRO  SAVIPL  DST=-(SP)
                MFPR    S^#PR$_IPL,DST
        .ENDM   SAVIPL

;
; SET PROCESSOR INTERRUPT PRIORITY LEVEL
;
; SETIPL IPL
;

        .MACRO  SETIPL IPL
                .IF NB  IPL
                MTPR    IPL,S^#PR$_IPL
                .IFF
                MTPR    #31,S^#PR$_IPL
                .ENDC
        .ENDM   SETIPL

;
; INITIATE SOFTWARE INTERRUPT
;
; SOFTINT IPL
;

        .MACRO  SOFTINT IPL
                MTPR    IPL,S^#PR$_SIRR
        .ENDM   SOFTINT
;++
; Macro to wait for a specific bit to become set/clear within a
```

```
; specified interval of time.  Uses a processor specific value
; established by system bootstrap to determine an approximate interval
; of time to wait instead of reading the processor clock.
;
; INPUTS:
;
;       TIME - the number of 10 micro-second intervals to wait
;       BITVAL - value of the bit(s) to test, i.e., the operand
;                specifier of the mask for a BITx instruction
;       SOURCE - the source operand specifier of the location to test
;       CONTEXT - either a "B", "W", or "L" specifying the context of
;                 the reference to the source operand
;       SENSE - whether to test fo the bit to be set or for it to be
;               cleared.  Devault (blank) is for set.  Else, specify
;               ".TRUE." or ".FALSE."
;
; OUTPUTS:
;
;       R0 - indicates success of failure status.  Success is defined as
;               the bit being at the specified sense within the specified
;               time interval.
;       R1 - destroyed, all other registers preserved.
;
;--

        .MACRO  TIMEWAIT  TIME,BITVAL,SOURCE,CONTEXT,SENSE,?L1,?L2,?L3

        MOVZWL  #SS$_NORMAL,R0          ; Assume success
        MULL3   TIME,G^EXE$GL_TENUSEC,R1; Calculate the time interval count
        CLRL    -(SP)                   ; Reserve space for delay loop index.
L1:     BIT'CONTEXT'    BITVAL,SOURCE   ; Test the bit

        .IF BLANK SENSE
        BNEQ    L2                      ; Conditionally branch on sense
        .IF_FALSE
        .IF IDENTICAL SENSE .TRUE.
        BNEQ    L2
        .IF_FALSE
        BEQL    L2
        .ENDC
        .ENDC

        MOVL    G^EXE$GL_UBDELAY,(SP)   ; Iteration count for delay loop.
L3:     SOBGTR  (SP),L3                 ; Delay loop to slow bit tests down
                                        ; to allow Unibus DMA to occur while
                                        ; testing a device register.
        SOBGTR  R1,L1                   ; Decrement interval count
        CLRL    R0                      ; Count expired, return failure
L2:
        TSTL    (SP)+                   ; Pop delay loop index off stack.
        .ENDM

;++
; TIMEDWAIT - Timed Wait Loop with Imbedded Tests
;
; Macro to wait for a specified interval of time.  Uses a processor
```

```
; specific value established by system bootstrap to determine an
; approximate interval of time to wait instead of reading the
; processor clock.  Instructions that test for various exit conditions
; may be imbedded within the wait loop, if so desired.
;
; INPUTS:
;
;       TIME - the number of 10 micro-second intervals to wait
;       INS1 - first instruction to imbed within wait loop
;       INS2 - second instruction to imbed within wait loop
;       INS3 - third instruction to imbed within wait loop
;       INS4 - fourth instruction to imbed within wait loop
;       INS5 - fifth instruction to imbed within wait loop
;       INS6 - sixth instruction to imbed within wait loop
;       DONELBL - label for exit from wait loop
;       IMBEDLBL - Label for imbedded instructions in wait loop
;       UBLBL - Label for UNIBUS SOBGTR loop
;
; OUTPUTS:
;
;       R0 - indicates success of failure status.  Success is defined as
;               the bit being at the specified sense within the specified
;               time interval.
;       R1 - destroyed, all other registers preserved.
;
;--

        .MACRO  TIMEDWAIT TIME,INS1,INS2,INS3,INS4,INS5,INS6,DONELBL,?IMBEDLBL,?UBLBL

        .nlist cnd
        MOVZWL  #SS$_NORMAL,R0          ; Assume success
        MULL3   TIME,G^EXE$GL_TENUSEC,R1; Calculate the time interval count
        CLRL    -(SP)                   ; Reserve space for delay loop index.
IMBEDLBL:
        'INS1'
        'INS2'
        'INS3'
        'INS4'
        'INS5'
        'INS6'
        MOVL    G^EXE$GL_UBDELAY,(SP)   ; Iteration count for delay loop.
UBLBL:  SOBGTR  (SP),UBLBL              ; Delay loop to slow bit tests down
                                        ; to allow Unibus DMA to occur while
                                        ; testing a device register.
        SOBGTR  R1,IMBEDLBL             ; Decrement interval count
        CLRL    R0                      ; Count expired, return failure
        .IF     NOT_BLANK, DONELBL
DONELBL:
        .ENDC
        TSTL    (SP)+                   ; Pop delay loop index off stack.
        .ENDM

; WAITFOR INTERRUPT OR TIMEOUT AND KEEP CHANNEL
;
; WFIKPCH EXCPT,TIME
;
```

```
        .MACRO  WFIKPCH EXCPT,TIME
                .IF B   TIME
                ASHL    #16,#1,-(SP)
                .IFF
                PUSHL   TIME
                .ENDC
                JSB     G^IOC$WFIKPCH
                .WORD   EXCPT-.
        .ENDM   WFIKPCH

;
; WAITFOR INTERRUPT OR TIMEOUT AND RELEASE CHANNEL
;
; WFIRLCH EXCPT,TIME
;
        .MACRO  WFIRLCH EXCPT,TIME
                .IF B   TIME
                ASHL    #16,#1,-(SP)
                .IFF
                PUSHL   TIME
                .ENDC
                JSB     G^IOC$WFIRLCH
                .WORD   EXCPT-.
        .ENDM   WFIRLCH
;
; System Communications Services (SCS) Macros
;
; ACCEPT - Accept a connection request
;
        .MACRO  ACCEPT, MSGADR=0,DGADR=0,ERRADR,INITCR=#0,MINSCR=#0, -
                        INITDG=#0,BLKPRI=#0,CONDAT=0,AUXSTR=0,BADRSP=0,?RETADR
                PUSHAB  B^RETADR
        $PUSHADR BADRSP
        $PUSHADR AUXSTR
        $PUSHADR CONDAT
                MOVZBW  BLKPRI,-(SP)
                MOVW    INITDG,-(SP)
                MOVW    MINSCR,-(SP)
                MOVW    INITCR,-(SP)
        .IF B ERRADR
                .ERROR  99      ; Error address parameter is required ;
        .IFF
                PUSHAB  ERRADR
        .ENDC
        $PUSHADR DGADR
        $PUSHADR MSGADR
                .GLOBAL   SCS$ACCEPT
                JMP     G^SCS$ACCEPT
RETADR:
        .ENDM   ACCEPT
;
; ALLOC_DG_BUF - Allocate a datagram buffer
;
        .MACRO  ALLOC_DG_BUF
```

```
                JSB     @PDT$L_ALLOCDG(R4)
        .ENDM   ALLOC_DG_BUF
;
; ALLOC_MSG_BUF - Allocate a message  buffer
;
        .MACRO  ALLOC_MSG_BUF
                JSB     @PDT$L_ALLOCMSG(R4)
        .ENDM   ALLOC_MSG_BUF
;
; ALLOC_RSPID - Allocate a response id
;
        .MACRO  ALLOC_RSPID
                JSB     G^SCS$ALLOC_RSPID
        .ENDM   ALLOC_RSPID
;
; CONFIG_PTH - Get path configuration information
;
        .MACRO  CONFIG_PTH,STAADR=0,OUTBUF=0
       $MOVEADR STAADR, R1
       $MOVEADR OUTBUF, R2
                JSB     G^SCS$CONFIG_PTH
        .ENDM   CONFIG_PTH
;
; CONFIG_SYS - Get system configuration information
;
        .MACRO  CONFIG_SYS,SYSADR=0,OUTBUF=0
       $MOVEADR SYSADR, R1
       $MOVEADR OUTBUF, R2
                JSB     G^SCS$CONFIG_SYS
        .ENDM   CONFIG_SYS
;
; CONNECT - Initiate a virtual circuit conncection
;
        .MACRO  CONNECT,MSGADR=0,DGADR=0,ERRADR,RSYSID=0,RSTADR=0,-
                RPRNAM=0,LPRNAM=0,INITCR=#0,MINSCR=#0,INITDG=#0,-
                BLKPRI=#0,CONDAT=0,AUXSTR=0,BADRSP=0,?RETADR
                PUSHAB  B^RETADR
       $PUSHADR BADRSP
       $PUSHADR AUXSTR
       $PUSHADR CONDAT
                MOVZBW  BLKPRI,-(SP)
                MOVW    INITDG,-(SP)
                MOVW    MINSCR,-(SP)
                MOVW    INITCR,-(SP)
       $PUSHADR LPRNAM
       $PUSHADR RPRNAM
       $PUSHADR RSTADR
       $PUSHADR RSYSID
       .IF B ERRADR
                .ERROR  99      ; Error address parameter is required ;
       .IFF
                PUSHAB  ERRADR
       .ENDC
       $PUSHADR DGADR
       $PUSHADR MSGADR
                JMP     G^SCS$CONNECT
```

```
RETADR:
        .ENDM   CONNECT
;
; DEALLOC_DG_BUF - Deallocate a datagram buffer
;
        .MACRO  DEALLOC_DG_BUF
                JSB     @PDT$L_DEALLOCDG(R4)
        .ENDM   DEALLOC_DG_BUF
;
; DEALLOC_MSG_BUF - Deallocate a message  buffer
;
        .MACRO  DEALLOC_MSG_BUF
                JSB     @PDT$L_DEALLOMSG(R4)
        .ENDM   DEALLOC_MSG_BUF
;
; DEALLOC_MSG_BUF_REG - Deallocate a message buffer
;
        .MACRO  DEALLOC_MSG_BUF_REG
                JSB     @PDT$L_DEALRGMSG(R4)
        .ENDM   DEALLOC_MSG_BUF_REG
;
; DEALLOC_RSPID - Deallocate a response id
;
        .MACRO  DEALLOC_RSPID
                JSB     G^SCS$DEALL_RSPID
        .ENDM   DEALLOC_RSPID
;
; DISCONNECT - Break a virtual circuit
;
        .MACRO  DISCONNECT,DISTYP
         .IF NB DISTYP
                MOVL    DISTYP, R0
         .ENDC
                JSB     G^SCS$DISCONNECT
        .ENDM   DISCONNECT
;
; FIND_RSPID_RDTE - Locate and validate the RDTE for a given response ID
;
        .MACRO  FIND_RSPID_RDTE
                JSB     G^SCS$FIND_RDTE
        .ENDM   FIND_RSPID_RDTE
;
; LISTEN - Listen for incoming CONNECT requests
;
        .MACRO  LISTEN,MSGADR=0,ERRADR,LPRNAM=0,PRINFO=0,?RETADR
                PUSHAB  B^RETADR
        $PUSHADR PRINFO
        $PUSHADR LPRNAM
        .IF B ERRADR
                .ERROR  99      ; Error address parameter is required ;
        .IFF
                PUSHAB  ERRADR
        .ENDC
        $PUSHADR MSGADR
                .GLOBAL   SCS$LISTEN
                JMP     G^SCS$LISTEN
```

```
RETADR:
        .ENDM   LISTEN

;+
; LOADVEC - conditionally defines a vector or a relative offset.
;
;       TYPE    = Type of vector (or offset) to create.
;                 Valid types are:
;                       1 : aligned longword of data
;                       2 : aligned JMP
;                       3 : unaligned JMP
;                       4 : specified data
;                       5 : specified JMP
;
;       ENTRY   = Entry point label of the routine to be loaded.  If
;                 PRMSW is not defined, a vector with this label will
;                 be defined in system space.
;
;       DEF_RTN = Address of a default routine.  This address is the
;                 initial target of the JMP vector.  This address is
;                 replaced with the actual routine address when the
;                 code is loaded (by EXE$LINK_VEC).
;
;       SEC_LABEL=Label within the code if different from the SYS
;                 entry name.  (Required for types 4 and 5).
;-
        .MACRO  LOADVEC ENTRY,TYPE=3,DEF_RTN=EXE$LOAD_ERROR,SEC_LABEL

        .IF     LE,TYPE                 ; Check for valid TYPE code
         .ERROR                         ; Illegal value;  1 <= VALUE <= 3
        .IFF
         .IF    GT,TYPE-5
          .ERROR                        ; Illegal value;  1 <= VALUE <= 3
         .ENDC
        .ENDC

        .IF     NDF,PRMSW               ; For linkage with SYS.EXE,...
         ;
         ; Handle type 1, longword data items
         ;
         .IF    EQ,TYPE-1
          .ALIGN LONG
ENTRY::                                 ; Define system vector
          .LONG  0
         .ENDC

         ;
         ; Handle type 2, aligned JMP
         ;
         .IF    EQ,TYPE-2
          .ALIGN LONG
ENTRY::                                 ; Define system vector
          JMP   @#'DEF_RTN
         .ENDC
```

```
;
; Handle type 3, unaligned JMP
;
.IF     EQ,TYPE-3
ENTRY::                                 ; Define system vector
 JMP    @#'DEF_RTN
.ENDC

;
; Handle type 4, specified Data
;
.IF     EQ,TYPE-4
 .ALIGN LONG
ENTRY::                                 ; Define system vector
 .LONG  0
.ENDC

;
; Handle type 5, specified jump
;
.IF     EQ,TYPE-5
ENTRY::                                 ; Define system vector
 JMP    @#'DEF_RTN
.ENDC


.IFF                                    ; For linkage with loadable code
 .IF    LE,TYPE-3                       ; (for types = 1,2,3)
  .BYTE  TYPE
   .IF BLANK SEC_LABEL
    .LONG  <ENTRY-.>
   .IFF
    .LONG  <SEC_LABEL-.>
   .ENDC
 .IFF
  .IF   LE,TYPE-5                       ; For linkage with loadable code and
   .BYTE TYPE                           ; SYS.STB (For types = 4,5)
   .ADDRESS ENTRY
   .LONG   <SEC_LABEL-.>
  .ENDC
 .ENDC
.ENDC

.ENDM


;
; MAP - Map a buffer for block transfer
;
.MACRO  MAP
        JSB     @PDT$L_MAP(R4)
.ENDM   MAP
;
; MAP_BYPASS - Map a buffer for block transfer and bypass
;
```

```
        .MACRO  MAP_BYPASS
                JSB     @PDT$L_MAPBYPASS(R4)
        .ENDM   MAP_BYPASS
;
; MAP_IRP - Map a buffer for block transfer, extract
;
        .MACRO  MAP_IRP
                JSB     @PDT$L_MAPIRP(R4)
        .ENDM   MAP_IRP
;
; MAP_IRP_BYPASS - Map a buffer for block transfer, extract
;
        .MACRO  MAP_IRP_BYPASS
                JSB     @PDT$L_MAPIRPBYP(R4)
        .ENDM   MAP_IRP_BYPASS
;
; MRESET - Reset remote port and system
;
        .MACRO  MRESET,RSTADR,FLAG=#0
                MOVL    FLAG,R0
        $MOVEADR RSTADR,R1
                JSB     @PDT$L_MRESET(R4)
        .ENDM   MRESET
;
; MSTART - Start remote port and system
;
        .MACRO  MSTART,RSTADR,FLAG=#1,START=#0
                MOVL    FLAG,R0
        $MOVEADR RSTADR,R1
                MOVL    START,R2
                JSB     @PDT$L_MSTART(R4)
        .ENDM   MSTART
;
; QUEUE_MULT_DGS - Add or subtract buffers from the datagram
;
        .MACRO  QUEUE_MULT_DGS,NUMBUF
         .IF NB NUMBUF
                MOVL    NUMBUF, R1
         .ENDC
                JSB     @PDT$L_QUEUEMDGS(R4)
        .ENDM   QUEUE_MULT_DGS
;
; QUEUE_DG_BUF - Queue a datagram buffer for receive
;
        .MACRO  QUEUE_DG_BUF
                JSB     @PDT$L_QUEUEDG(R4)
        .ENDM   QUEUE_DG_BUF
;
; READ_COUNTERS - Read and initialize port counters
;
        .MACRO  READ_COUNTERS,RSTADR=0,LPRNAM
        $MOVEADR RSTADR,R0
        $MOVEADR LPRNAM,R1
                JSB     @PDT$L_READCOUNT(R4)
        .ENDM   READ_COUNTERS
;
```

```
;
; RECYCL_MSG_BUF - Recycle a message buffer,low
;
        .MACRO  RECYCL_MSG_BUF
                JSB     @PDT$L_RCLMSGBUF(R4)
        .ENDM   RECYCL_MSG_BUF
;
; RECYCH_MSG_BUF - Recycle a message buffer,high
;
        .MACRO  RECYCH_MSG_BUF
                JSB     @PDT$L_RCHMSGBUF(R4)
        .ENDM   RECYCH_MSG_BUF
;
; RECYCL_RSPID - Recycle a response ID
;
        .MACRO  RECYCL_RSPID
                JSB     G^SCS$RECYL_RSPID
        .ENDM   RECYCL_RSPID
;
; REJECT - Reject a connection request
;
        .MACRO  REJECT,REJTYP
         .IF NB REJTYP
                MOVL    REJTYP, R0
         .ENDC
                JSB     @PDT$L_REJECT(R4)
        .ENDM   REJECT
;
; REQUEST_DATA - Request block transfer data
;
        .MACRO  REQUEST_DATA,?L1
                JSB     G^SCS$ALLOC_RSPID
                JSB     @PDT$L_ALLOCMSG(R4)
                BLBC    R0,L1
                JSB     @PDT$L_REQDATA(R4)
        L1:
        .ENDM   REQUEST_DATA
;
; RLS_COUNTERS - Release counters
;
        .MACRO  RLS_COUNTERS
                JSB     @PDT$L_RLSCOUNT(R4)
        .ENDM   RLS_COUNTERS
;
; SCAN_MSGBUF_WAIT - Scan message buffer and send credit wait queues for
;       CDRP with given CDT
;
        .MACRO  SCAN_MSGBUF_WAIT,ACTION
                MOVAB   ACTION, R0
                JSB     G^SCS$LKP_MSGWAIT
        .ENDM   SCAN_MSGBUF_WAIT
;
; SCAN_RDT - Scan RDT for CDRP with given CDT
;
        .MACRO  SCAN_RDT,ACTION
                MOVAB   ACTION, R0
                JSB     G^SCS$LKP_RDTCDRP
```

```
        .ENDM   SCAN_RDT
;
; SCAN_RSPID_WAIT - Scan RSPID wait queue for CDRP with given CDT
;
        .MACRO  SCAN_RSPID_WAIT,ACTION
                MOVAB   ACTION, R0
                JSB     G^SCS$LKP_RDTWAIT
        .ENDM   SCAN_RSPID_WAIT
;
; SEND_DATA - Send block transfer data
;
        .MACRO  SEND_DATA,?L1
                JSB     G^SCS$ALLOC_RSPID
                JSB     @PDT$L_ALLOCMSG(R4)
                BLBC    R0,L1
                JSB     @PDT$L_SENDDATA(R4)
        L1:
        .ENDM   SEND_DATA
;
; SEND_DG_BUF - Send a datagram
;
        .MACRO  SEND_DG_BUF,FLAG
         .IF NB FLAG
                MOVL    FLAG, R0
         .ENDC
        JSB     @PDT$L_SENDDG(R4)
        .ENDM   SEND_DG_BUF
;
; SEND_DG_BUF_REG - Send a datagram without a CDRP.
;
        .MACRO  SEND_DG_BUF_REG,FLAG,CDT=,BUFFER=,SIZE=
         .IF NB FLAG
                MOVL    FLAG,R0
         .ENDC
         .IF NB CDT
                MOVL    CDT,R3
         .ENDC
         .IF NB BUFFER
                MOVL    BUFFER,R2
         .ENDC
         .IF NB SIZE
                MOVL    SIZE,R1
         .ENDC
        JSB     @PDT$L_SENDRGDG(R4)
        .ENDM   SEND_DG_BUF_REG
;
; SEND_CNT_MSG_BUF - Send a message with byte count
;
        .MACRO  SEND_CNT_MSG_BUF
                JSB     @PDT$L_SNDCNTMSG(R4)
        .ENDM   SEND_CNT_MSG_BUF
;
; SEND_MSG_BUF - Send a message
;
        .MACRO  SEND_MSG_BUF
                JSB     @PDT$L_SENDMSG(R4)
```

```
        .ENDM   SEND_MSG_BUF
;
; UNMAP - Unmap a buffer for block transfer
;
        .MACRO  UNMAP
                JSB     @PDT$L_UNMAP(R4)
        .ENDM   UNMAP
;
; Macros for invocation of Machine Check recovery blocks
;
; $PRTCTINI - set start of recovery block
;       LABEL = end of recovery block label (must be same label as $PRTCTEND)
;       MASK = bit mask for types of errors to protect against
;
        .MACRO  $PRTCTINI LABEL,MASK
                PUSHAL  LABEL
                MOVL    MASK,R0
                JSB     G^EXE$MCHK_PRTCT
        .ENDM

;
; $PRTCTEND - macro for defining end of current recovery block.
;       LABEL = end of recovery block label (must be same as in $PRTCTINI)
;
        .MACRO  $PRTCTEND LABEL
                RSB
LABEL:
        .ENDM
;
; $PRTCTEST - test to see if recovery block in effect for current error
;       ADDRESS = pointer to PC,PSL pair of error interrupt on stack
;       MASK = bits defining type of error
;
        .MACRO  $PRTCTEST ADDRESS,MASK
                MOVAL   ADDRESS,R1
                MOVL    MASK,R2
                JSB     G^EXE$MCHK_TEST
        .ENDM

;
; $BUGPRTCT - Macro to test whether or not recovery block in effect
;               for this BUGCHECK
;               Arguments already on current (Interrupt) stack
;
        .MACRO  $BUGPRTCT
                JSB     G^EXE$MCHK_BUGCHK
        .ENDM


;
; SYSTEM LOADABLE VECTOR TABLE
;
        SLVTAB  END,INITRTN,SUBTYP,PROT_R,PROT_W,FACILITY
;
        END     =       Address at end of image
        INITRTN =       Address of Initialization Routine
```

```
;       SUBTYP  =       Sub type of image
;       PROT_R  =       Page protection to be applied to writeable image
;       PROT_W  =       Page protection to be applied to read-only image
;       FACILITY=       facility name
;       SYSVECS =       Address of vectors in SYS.EXE
;
;
        .MACRO  SLVTAB  END,INITRTN,SUBTYP=0,PROT_R,PROT_W,FACILITY,SYSVECS,?L1,?L2
        $DYNDEF
        $PRTDEF
L1:
        .LONG   END-L1

        .IF NB INITRTN
        .LONG    INITRTN-L1
        .IFF
        .LONG   0
        .ENDC

        .WORD   END-L1
        .BYTE   DYN$C_LOADCODE
        .BYTE   SUBTYP

        .IF NB PROT_R
        .BYTE    PROT_R
        .IFF
        .BYTE   PRT$C_ER
        .ENDC

        .IF NB PROT_W
        .BYTE    PROT_W
        .IFF
        .BYTE   PRT$C_EW
        .ENDC

        .WORD   0

        .IF NB SYSVECS
        .ADDRESS SYSVECS
        .IFF
        .LONG   0
        .ENDC

L2:     .ASCIC  /FACILITY/
        .=L2+16

        .MDELETE SLVTAB
        .ENDM   SLVTAB

;
; TEST WHETHER THIS SYSTEM IS A MEMBER OF A CLUSTER AND
; BRANCH IF IT IS A MEMBER
;
; IFCLSTR       DEST
;
```

```
        .MACRO  IFCLSTR DEST
                TSTL    G^CLU$GL_CLUB
                BNEQ    DEST
        .ENDM   IFCLSTR


;
; TEST WHETHER THIS SYSTEM IS A MEMBER OF A CLUSTER AND
; BRANCH IF IT IS NOT A MEMBER
;
; IFNOCLSTR     DEST
;

        .MACRO  IFNOCLSTR DEST
                TSTL    G^CLU$GL_CLUB
                BEQL    DEST
        .ENDM   IFNOCLSTR


;
; Macros to allow declaration of Adapter types and Adapter initialization
; routines.  These macros are meant to be invoked in modules that are linked
; into SYSLOAXXX.EXE images.
;

;++
; Macro ADAPDESC.
;       Create NUM_PAGES, INIT_ROUTINES, and ADAPTERS arrays.
; INPUTS:
;       ADPTYPES - List of specific nexus device (adapter) types that conform
;                  to the general type described by the remainder of the input
;                  arguments.
;       NUMPAGES - The number of pages required for the adapter's register
;                  space.
;       INITRTN - The name of an adapter-specific initialization routine.
;
; Note: Each invocation of this macro corresponds to 1 "general" adapter type.
;       Each element in an ADPTYPES list corresponds to 1 "specific" type.
;
; Note: This macro is invoked in one of two environments.  These environments
;       are defined by whether or not the symbol $$$VMSDEFINED is defined or
;       not.  When the symbol is defined, this means that we are expanding an
;       invocation of the macro supported by DIGITAL, appearing in the module
;       INIADPxxx, whereas, if the symbol is NOT defined, this is a user
;       written invocation.  The only difference in the compiled data is that
;       a separate set of .PSECT's are used for the two environments.
;--
        .MACRO  ADAPDESC        ADPTYPES,NUMPAGES,INITRTN
        .SAVE

;
; Create three arrays; a list of specific device type codes (NDT$_),
; a NUM_PAGES array that contains the number of pages to be mapped for each
; corresponding device types, and the INIT_ROUTINES array that contains
; self relative addresses of initialization routines for the corresponding
; device types.  Each array is contained in two .PSECTs, with the first
```

```
; such .PSECT of a pair, containing DIGITAL contributions and the second
; .PSECT containing user contributions.
;
        .IRP    ADPTYPE,ADPTYPES        ; Repeat for each unique adp type...

        .if     DF      $$$VMSDEFINED   ; If VMS distributed software.
                .PSECT  $$$INIT$DATA0
        .iff                            ; If user written invocation.
                .PSECT  $$$INIT$DATA1
        .endc                           ; End .PSECT selection conditional.
        .LONG   ADPTYPE                 ; Add an entry to ADAPTERS array.

        .if     DF      $$$VMSDEFINED   ; If VMS distributed software.
                .PSECT  $$$INIT$DATA2   ; Add an entry to the NUM_PAGES array.
        .iff                            ; If user written invocation.
                .PSECT  $$$INIT$DATA3   ; Add an entry to the NUM_PAGES array.
        .endc                           ; End .PSECT selection conditional.
        .WORD   NUMPAGES                ; Store number of pages to be mapped.

        .if     DF      $$$VMSDEFINED   ; If VMS distributed software.
                .PSECT  $$$INIT$DATA4   ; Add entry to the INIT_ROUTINES array.
        .iff                            ; If user written invocation.
                .PSECT  $$$INIT$DATA5   ; Add entry to the INIT_ROUTINES array.
        .endc                           ; End .PSECT selection conditional.
        .IF NOT_BLANK INITRTN           ; If ADP init routine is specified...
                .LONG   <INITRTN-.>     ; Add self-relative pointer to routine.
        .IF_FALSE                       ; Else...
                .LONG   0               ; Add a 0 entry to INIT_ROUTINES.
        .ENDC

        .ENDR

        .RESTORE
        .ENDM   ADAPDESC

;
; ADAP_INIRUT - macro to declare label of an adapter initialization routine
; and to set the proper .PSECT so that the routine will be properly placed
; when linked into SYSLOAXXX.EXE.
;

        .MACRO  ADAP_INIRUT     ROUTINE

        .PSECT  $$$INIT$CODE,QUAD
ROUTINE.
        .ENDM   ADAP_INIRUT
        .LIST
```

ALLOCPFN
LIS

ACCOUNT
LIS

CMODSSDSP
MAR

SYSMAR
MAR

SYSPARAM
MAR

ASTDEL
LIS